APPENDIX A:

The following is a header file for the interprocess communication (IPC) feature of MMLite. This is the interface for the loadable IPC. IPC.EXE registers an object that obeys this interface in the namespace, at the time IPC.EXE is loaded (on demand). The method of interest is the TrapHandler() one, which is passed a pointer (pThreadState) to the machine-level register state of the thread. The method must infer from the state what is the operation the application intends to perform. For instance, some of the operations that must be implemented by IPC.EXE are:

- creation and destruction of endpoints
- linking of endpoints (export/import)
- remote method invocation.

We do not specify these, they are specific of the particular style of IPC. For example, there might be a special value in one of the machine registers that indicates what operation is desired.

```
#if defined(__cplusplus)
extern "C" {
#endif /* __cplusplus */

typedef interface ILoadableIPC ILoadableIPC;

    typedef struct ILoadableIPCVtbl {
     SCODE ( MCT *QueryInterface )(
         ILoadableIPC *This,
         /* [in] */ REFIID Iid,
         /* [out] */ void **ppObject);

     UINT ( MCT *AddRef )(
```

```
            ILoadableIPC *This);

        UINT ( MCT *Release )(
            ILoadableIPC *This);

        SCODE ( MCT *TrapHandler )(
            ILoadableIPC *This,
               PCXTINFO pThreadState
               );

    } ILoadableIPCVtbl;

#if defined(__cplusplus) && !defined(CINTERFACE)

    interface ILoadableIPC : public IUnknown
    {
    public:
     virtual SCODE MCT TrapHandler(
               PCXTINFO pThreadState
               ) = 0;

    };

#else  /* __cplusplus */  /* C style interface */

    interface ILoadableIPC
    {
     CONST_VTBL struct ILoadableIPCVtbl *lpVtbl;
    };

#endif  /* __cplusplus */ /* C style interface */

#if defined(__cplusplus)
}
```

```
#endif /* __cplusplus */
```

APPENDIX B:

The following is a header file for the mutation (Imutate) feature of MMLite. This is the interface to the object mutation algorithm. The mutator thread invokes one of the two methods MutateObject or MutateVtable. The first is the general case, the second is a special case where the object state does not need to actually change. There might be other special cases, such as mutating a single method. The general case covers them all, it just might be less efficient. This interface is registered in the namespace by the component that implements the mutation algorithm(s). This is a loadable component.

```
#if defined(__cplusplus)
extern "C" {
#endif /* __cplusplus */

typedef BOOL (MCT *TRANSITION_FUNCTION)(
        IUnknown *pObj
    );

typedef interface IMutate IMutate;

typedef UINT MUTATE_FLAGS;
#define MUTATE_AS_APPLICABLE (0x0)
#define MUTATE_BY_MUTEX (0x1)
#define MUTATE_BY_ABORT (0x2)
#define MUTATE_BY_SWIZZLING (0x3)
#define MUTATE_BY_STOMPING (0x4)

    typedef struct IMutateVtbl {
```

```
        SCODE ( MCT *QueryInterface )(
            IMutate *This,
            /* [in] */ REFIID Iid,
            /* [out] */ void **ppObject);

        UINT ( MCT *AddRef )(
            IMutate *This);

        UINT ( MCT *Release )(
            IMutate *This);

        SCODE ( MCT *MutateVtable )(
            IMutate *This,
            IUnknown *pWho,
               struct IUnknownVtbl *pNewVtable,
               MUTATE_FLAGS dwFlags);

        SCODE ( MCT *MutateObject )(
            IMutate *This,
            IUnknown *pWho,
               TRANSITION_FUNCTION pTransition,
               MUTATE_FLAGS dwFlags);

    } IMutateVtbl;

#if defined(__cplusplus) && !defined(CINTERFACE)

    interface IMutate : public IUnknown
    {
    public:
     virtual SCODE MCT MutateVtable(
        IUnknown *pWho,
           struct IUnknownVtbl *pNewVtable,
           MUTATE_FLAGS dwFlags) = 0;
```

```
        virtual SCODE MCT MutateObject(
            IUnknown *pWho,
               TRANSITION_FUNCTION pTransition,
               MUTATE_FLAGS dwFlags) = 0;

        };

    #else  /* __cplusplus */  /* C style interface */

        interface IMutate
        {
         CONST_VTBL struct IMutateVtbl *lpVtbl;
        };

    #endif  /* __cplusplus */ /* C style interface */

    #if defined(__cplusplus)
    }
    #endif /* __cplusplus */
```

Appendix C:

The following is a header file of the virtual memory manager of MMLite. These are the 3 interfaces to the loadable virtual memory (VM). [The IVmMapping interface is referred to above as VMMap.] The fourth interface (VMFactory) is the one that is actually registered in the namespace; it contains the constructors for objects of the other 3 types.

```
#ifndef _MMVM_H_
```

```
#define _MMVM_H_

typedef struct IUnknown *PIUNKNOWN;
typedef struct IFile *PIFILE;
typedef struct IVmView *PIVMVIEW;
typedef struct IVmSpace *PIVMSPACE;
typedef struct IVmMapping *PIVMMAPPING;

/* Basic types */
typedef UINT64 VM_OFFSET;
typedef UINT64 VM_SIZE;
typedef UINT32 VM_FLAGS;

/* Other necessary types */
typedef struct _IPAGELIST *PIPAGELIST;

/* VM_FLAGS bit definitions */
#define VM_PROT_READ          1
#define VM_PROT_WRITE         2
#define VM_PROT_EXECUTE       4
#define VM_COPYONWRITE        8
#define VM_WIRE            0x10    /* Lock memory in core */
#define VM_NOCACHE         0x20    /* Turn off hardware
cache, for i/o etc. */
#define VM_SHARE           0x40    /* Recursion: Share mem w
underlying VMSpace */
#define VM_PAGE_PRESENT 0x1000     /* The page is in core */
#define VM_PAGE_DIRTY   0x2000     /* The page has been
written and is writable */
#define VM_PAGE_BUSY     0x4000    /* Undergoing transition
and is unaccessible */
#define VM_PAGE_WAITING 0x8000     /* somebody is waiting
for the busy page */
```

```
#define VM_PAGE_VTLB    0x10000     /* page has been looked
up since VTLB_Flush */
#define VM_PAGE_MARK    0x20000     /* page has been touched
(for pseudo-LRU) */

#define VM_FLAGS_MASK     0x7f      /* Flags settable in vs-
>Protect. */

/* ------------------ Mapping -------------------------- */
/* A mapping object is used for providing backing storage
for an
 * address space.  A mapping object can map a file or
another address
 * space (or whatever you can think of).
 *
 * A mapping object has an offset into the backing store
object it maps,
 * a size, and a maximum protection.
 *
 * Data
 */
struct VmMappingVtbl {
    SCODE (*QueryInterface)(PIVMMAPPING This, REFIID Iid,
void **ppObject);
    UINT (*AddRef)(PIVMMAPPING This);
    UINT (*Release)(PIVMMAPPING This);

    /* Get pages from backing store. */
    SCODE (*Read)(PIVMMAPPING This, VM_OFFSET Offset,
VM_SIZE Size,
            PIPAGELIST *PageList);

    /* Write pages out into backing store */
```

```
    SCODE (*Write)(PIVMMAPPING This, VM_OFFSET Offset,
VM_SIZE Size,
                BOOL Consume, PIPAGELIST PageList);

    /* Like read but shares pages instead of copying them
(lookup func). */
    SCODE (*Share)(PIVMMAPPING This, VM_OFFSET Offset,
VM_SIZE Size,
                VM_FLAGS Access, PIPAGELIST *pPageList);

    /* Check protection of area in order to do checks at
vm_prot time.
     * Returns lowest possible access combination
(intersection).
     */
    SCODE (*QueryAccess)(PIVMMAPPING This, VM_OFFSET Offset,
VM_SIZE Size,
                    VM_FLAGS *pAccess);

    SCODE (*GetSize)(PIVMMAPPING This, VM_SIZE *pSize);

    /* Create a new mapping from this mapping with limited
access */
    SCODE (*Clone)(PIVMMAPPING This, VM_OFFSET Offset,
VM_SIZE Size,
                VM_FLAGS Access, PIVMMAPPING *pNewMapping);
};

struct IVmMapping {
    CONST_VTBL struct VmMappingVtbl *v;
};

extern const IID IID_IVmMapping;
```

```
/* ------------------ Adress Space ------------------------
- */
struct VMSpaceVtbl {
    SCODE (*QueryInterface)(PIVMSPACE This, REFIID Iid, void
**ppObject);
    UINT (*AddRef)(PIVMSPACE This);
    UINT (*Release)(PIVMSPACE This);

    SCODE (*Reserve)(PIVMSPACE This, VM_OFFSET Offset,
VM_SIZE Size);

    /* Delete mappings on range.  Also unreserve if flag set
*/
    SCODE (*Delete)(PIVMSPACE This, VM_OFFSET Offset,
VM_SIZE Size,
                BOOL Unreserve);

    /* Map backing store into address space.  CopyMapping is
used only
     * with VM_WRITECOPY.  The flags are ored with those
already in place,
     * thus it is ok to vs->Protect before mapping (XXX Is
this useful?).
     */
    SCODE (*Map)(PIVMSPACE This, VM_OFFSET Offset, VM_SIZE
Size,
            PIVMMAPPING Mapping, PIVMMAPPING CopyMapping,
VM_FLAGS Flags);

    /* Set protection or other permanent attributes */
    /* Recurse removes atomicity and affects only certain
attributes:
     * namely VM_WIRE, VM_NOCACHE.
```

```
     * Treat Attributes as mask to be set (Set=TRUE) or
unset (Set=FALSE).
     */
    SCODE (*Protect)(PIVMSPACE This, VM_OFFSET Offset,
5   VM_SIZE Size,
                VM_FLAGS Attributes, BOOL Set, BOOL Recurse);

    /* Set non-permanent attributes */
    #define VM_CACHE_CLEAN    1     /* write out dirty pages */
10  #define VM_CACHE_FLUSH    2     /* throw away pages after
    cleaning them */
    #define VM_CACHE_DISCARD 3     /* throw away all pages w/o
    cleaning */
    #define VM_CACHE_READ     4     /* emulate read fault */
15  #define VM_CACHE_WRITE    5     /* emulate write fault */
    #define VM_CACHE_COPY     6     /* realize copy on writes */
    #define VM_CACHE_UNMARK   7     /* Pseudo-LRU: mark pages
    unused */
    #define VM_CACHE_SWEEP    8     /* Pseudo-LRU: throw away
20  untouched pages */
                                    /* SWEEP is same as FLUSH
    for unmarked pages */
    SCODE (*CacheControl)(PIVMSPACE This, VM_OFFSET Offset,
    VM_SIZE Size,
25                  UINT Action, BOOL Recurse);

    /* vm_regions: Return info from region at or beyond
    offset */
    /* Besides This and pOffset the arguments may be NULL */
30  SCODE (*QueryVM)(PIVMSPACE This, VM_OFFSET
    *pOffset/*in/out*/,
            VM_SIZE *pSize, VM_FLAGS *pFlags,
            PIVMMAPPING *pMapping, PIVMMAPPING
    *pCopyMapping);
```

```
    /* Create New; move regs from This into New; map New
into This w/ Flags.
     * If Flags indicate COW then copies are paged to
CopyMapping.
     */
    SCODE (*CreateShadow)(PIVMSPACE This, VM_OFFSET Offset,
VM_SIZE Size,
                  PIVMMAPPING CopyMapping, VM_FLAGS Flags,
                  PIVMSPACE *pNew);
};

struct IVmSpace {
    CONST_VTBL struct VMSpaceVtbl *v;
};

extern const IID IID_IVmSpace;

/* ------------------- VMView interface --------------------
----- */
#if defined(__cplusplus) && !defined(CINTERFACE)
#else
typedef struct IVmView *PIVMVIEW;
typedef struct IVmMapping *PIVMMAPPING;

struct VMViewVtbl {
    SCODE (*QueryInterface)(PIVMVIEW This, REFIID Iid, void
**ppObject);
    UINT (*AddRef)(PIVMVIEW This);
    UINT (*Release)(PIVMVIEW This);

    /* SwitchTo is called from context switch path */
    SCODE (*SwitchTo)(PIVMVIEW This);
    /* Fault is called at pagefault time */
```

```
    SCODE (*Fault)(PIVMVIEW This, ADDRESS FaultAddr, UINT
Access);
    SCODE (*SetMapping)(PIVMVIEW This, PIVMMAPPING Mapping);
    SCODE (*GetMapping)(PIVMVIEW This, PIVMMAPPING
*pMapping);
};

struct IVmView {
    CONST_VTBL struct VMViewVtbl *v;
};
#endif /* __cplusplus */

extern const IID IID_IVmView;

/* -------------------- VM constructor interface ------------
------------- */
#if defined(__cplusplus) && !defined(CINTERFACE)
#else
typedef struct IVmFactory *PIVMFACTORY;

struct VmFactoryVtbl {
    SCODE (*QueryInterface)(PIVMFACTORY This, REFIID Iid,
void **ppObject);
    UINT (*AddRef)(PIVMFACTORY This);
    UINT (*Release)(PIVMFACTORY This);

    SCODE (*CreateVmView)(PIVMFACTORY This, PIVMVIEW *pVV);
    SCODE (*CreateVmSpace)(PIVMFACTORY This, PIVMSPACE
*pVS);
    SCODE (*CreateVmMappingFromFile)(PIVMFACTORY This,
PIFILE File, PIVMMAPPING *pNewMapping);
    SCODE (*CreateVmMappingFromZero)(PIVMFACTORY This,
PIVMMAPPING *pNewMapping);
```

```
    SCODE (*CreateDefault)(PIVMFACTORY This, ADDRESS Where,
VM_SIZE Size, PIVMVIEW *pVV);
};

struct IVmFactory {
    CONST_VTBL struct VmFactoryVtbl *v;
};
#endif /* __cplusplus */

extern const IID IID_IVmFactory;

#endif /* __MMVM_H_ */
```

APPENDIX D:

The following is a header file for the basic features of MMLite. This file contains all the basic interfaces and services of the MMLite "kernel", including the NameSpace. The NameSpace is the one that implements the on-demand loading programming

paradigm. The Register() method is used to add objects to the namespace and the
Bind() method is used to look them up. The IHeap is the interface that applications use to request memory allocations (with or without VM). There are also other functions that are not defined as part of a specific interface. Among these, AddDevice() is the one used to install the VM and IPC trap handlers.

As for additional interfaces, the VM manager uses the IVTLB interface. The primitives defined in mmbase.h (for Mutex_* and Condition_*, and for constraints) are sufficient in the general case.

```
#ifndef _MMBASE_H_
#define _MMBASE_H_

/************* Constraint ****************/
#define CRITICAL    ( 1 )         /* size is 4 */
#define NONCRITICAL ( 0 )         /* size is 4 */
typedef /* [transmit] */ INT CRITICALITY;        /* size is
4 */

#if defined(__cplusplus)
/* work-around VC++ bug that produces warning 4705 */
typedef struct _TIME_CONSTRAINT {
    inline _TIME_CONSTRAINT() {};
    inline ~_TIME_CONSTRAINT() {};
    TIME        Start;
    TIME        Estimate;
    TIME        Deadline;
    CRITICALITY Criticality;
} TIME_CONSTRAINT, *PTIME_CONSTRAINT;
```

```
#else
typedef struct  _TIME_CONSTRAINT       /* size is 28 */
    {
    TIME Start;
    TIME Estimate;
    TIME Deadline;
    CRITICALITY Criticality;
    }   TIME_CONSTRAINT;

     /* size is 4 */
typedef struct _TIME_CONSTRAINT *PTIME_CONSTRAINT;
#endif /* __cplusplus */

/************* Mutex ***************/
#if defined(__cplusplus)
/* work-around VC++ bug that produces warning 4705 */
typedef struct _MUTEX {
    inline _MUTEX() {};
    inline ~_MUTEX() {};
    UINT _mutex_state[MUTEX_STATE_SIZE];
} MUTEX, *PMUTEX;
#else
typedef struct _MUTEX {      /* size is 4 */
    UINT _mutex_state[ 1 ];
} MUTEX;

typedef struct _MUTEX *PMUTEX;         /* size is 4 */
#endif /* __cplusplus */

/************* Condition ***************/
#if defined(__cplusplus)
/* work-around VC++ bug that produces warning 4705 */
typedef struct _CONDITION {
    inline _CONDITION() {};
```

```
        inline ~_CONDITION() {};
        UINT _cond_state[CONDITION_STATE_SIZE];
    } CONDITION, *PCONDITION;
    #else
 5  typedef struct _CONDITION {       /* size is 8 */
        UINT _cond_state[ 2 ];
    } CONDITION;

    typedef struct _CONDITION *PCONDITION;      /* size is 8 */
10  #endif /* __cplusplus */

    typedef interface IProcess IProcess;
    typedef interface IThread IThread;


15

    /******** namesp.h ********/

    #if defined(__cplusplus)
    extern "C" {
20  #endif /* __cplusplus */
    typedef interface INameSpace INameSpace;

    typedef UINT NAME_SPACE_FLAGS;
    #define NAME_SPACE_READ ( 0x1 )
25  #define NAME_SPACE_WRITE    ( 0x2 )
    #define NAME_SPACE_CREATE   ( 0x10 )
    #define NAME_SPACE_FAILIFEXIST  ( 0x20 )
    #define NAME_SPACE_EXTENSION    ( 0x800 )

30      typedef struct INameSpaceVtbl {
         SCODE ( MCT *QueryInterface )(
             INameSpace *This,
             /* [in] */ REFIID Iid,
             /* [out] */ void **ppObject);
```

```
        UINT ( MCT *AddRef )(
            INameSpace *This);

        UINT ( MCT *Release )(
            INameSpace *This);

        SCODE ( MCT *Register )(
            INameSpace *This,
            CTSTR pName,
            IUnknown *pObj,
            NAME_SPACE_FLAGS Flags,
            INameSpace *pServerNameSpace);

        SCODE ( MCT *Unregister )(
            INameSpace *This,
            CTSTR pName);

        SCODE ( MCT *Bind )(
            INameSpace *This,
            CTSTR pName,
            NAME_SPACE_FLAGS Flags,
            /* [out] */ IUnknown **ppUnk);

        SCODE ( MCT *FindFirst )(
            INameSpace *This,
            CTSTR pPrefix,
            /* [size_is][out] */ TSTR pBuffer,
            /* [in] */ UINT BufSize,
            /* [out] */ UINT *pStrLen);

        SCODE ( MCT *FindNext )(
            INameSpace *This,
            CTSTR pPrefix,
```

```
            CTSTR pPrevious,
            /* [size_is][out] */ TSTR pBuffer,
            /* [in] */ UINT BufSize,
            /* [out] */ UINT *pStrLen);

        SCODE ( MCT *GetCapabilities )(
            INameSpace *This,
            CTSTR pName,
            /* [out] */ NAME_SPACE_FLAGS *pFlags);

        } INameSpaceVtbl;

    #if defined(__cplusplus) && !defined(CINTERFACE)

        interface INameSpace : public IUnknown
        {
        public:
         virtual SCODE MCT Register(
            CTSTR pName,
            IUnknown *pObj,
            NAME_SPACE_FLAGS Flags,
            INameSpace *pServerNameSpace) = 0;

         virtual SCODE MCT Unregister(
            CTSTR pName) = 0;

         virtual SCODE MCT Bind(
            CTSTR pName,
            NAME_SPACE_FLAGS Flags,
            /* [out] */ IUnknown **ppUnk) = 0;

         virtual SCODE MCT FindFirst(
            CTSTR pPrefix,
            /* [size_is][out] */ TSTR pBuffer,
```

```
            /* [in] */ UINT BufSize,
            /* [out] */ UINT *pStrLen) = 0;

        virtual SCODE MCT FindNext(
            CTSTR pPrefix,
            CTSTR pPrevious,
            /* [size_is][out] */ TSTR pBuffer,
            /* [in] */ UINT BufSize,
            /* [out] */ UINT *pStrLen) = 0;

        virtual SCODE MCT GetCapabilities(
            CTSTR pName,
            /* [out] */ NAME_SPACE_FLAGS *pFlags) = 0;

    };

#else  /* __cplusplus */  /* C style interface */

    interface INameSpace
    {
     CONST_VTBL struct INameSpaceVtbl *lpVtbl;
    };

#endif  /* __cplusplus */ /* C style interface */

#if defined(__cplusplus)
}
#endif /* __cplusplus */
/******** heap.h ********/

#if defined(__cplusplus)
extern "C"{
#endif /* __cplusplus */
```

```
typedef interface IHeap IHeap;
#define HEAP_NO_SERIALIZE   ( 0x1 )
#define HEAP_NO_COPY (0x4)
#define HEAP_ZERO_MEMORY    ( 0x8 )
#define HEAP_REALLOC_IN_PLACE_ONLY  ( 0x10 )
#define HEAP_CACHE_ALIGN ( 0x100 )

typedef struct IHeapVtbl {
    SCODE ( MCT *QueryInterface )(
         IHeap *This,
         /* [in] */ REFIID Iid,
         /* [out] */ void **ppObject);

    UINT ( MCT *AddRef )(
         IHeap *This);

    UINT ( MCT *Release )(
         IHeap *This);

    PTR ( MCT *Alloc )(
         IHeap *This,
         UINT Flags,
         UINT Size,
         UINT Alignment);

    PTR ( MCT *ReAlloc )(
         IHeap *This,
         UINT Flags,
         PTR pMem,
         UINT NewSize,
         UINT Alignment);

    BOOL ( MCT *Free )(
         IHeap *This,
```

```
            UINT Flags,
            PTR pMem);

        ADDR_SIZE ( MCT *Size )(
            IHeap *This,
            UINT Flags,
            PTR pMem);

        BOOL ( MCT *Validate )(
            IHeap *This,
            UINT Flags,
            PTR pMem);

        PTR ( MCT *Extract )(
            IHeap *This,
            UINT Flags,
            PTR pMem,
            UINT Size);

        SCODE ( MCT *Status )(
            IHeap *This,
            ADDR_SIZE *pReserve,
            ADDR_SIZE *pCommit,
            ADDR_SIZE *pUsed,
            ADDR_SIZE *pMaxUsed);

    } IHeapVtbl;

    #if defined(__cplusplus) && !defined(CINTERFACE)
    interface IHeap : public IUnknown {
      public:
        virtual PTR MCT Alloc(
            UINT Flags,
            UINT Size,
```

```
            UINT Alignment) = 0;

        virtual PTR MCT ReAlloc(
            UINT Flags,
            PTR pMem,
            UINT NewSize,
            UINT Alignment) = 0;

        virtual BOOL MCT Free(
            UINT Flags,
            PTR pMem) = 0;

        virtual ADDR_SIZE MCT Size(
            UINT Flags,
            PTR pMem) = 0;

        virtual BOOL MCT Validate(
            UINT Flags,
            PTR pMem) = 0;

        virtual PTR MCT Extract(
            UINT Flags,
            PTR pMem,
            UINT Size) = 0;

        virtual SCODE MCT Status(
            ADDR_SIZE *pReserve,
            ADDR_SIZE *pCommit,
            ADDR_SIZE *pUsed,
            ADDR_SIZE *pMaxUsed) = 0;

    };
    #else   /* C style interface */
    interface IHeap {
```

```
        CONST_VTBL struct IHeapVtbl *lpVtbl;
    };
    #endif  /* __cplusplus */ /* C style interface */
    #if defined(__cplusplus)
 5  }
    #endif /* __cplusplus */
    /******** module.h ********/
    #if defined(__cplusplus)
    extern "C"{
10  #endif /* __cplusplus */

    typedef interface IModule IModule;
    typedef struct IModuleVtbl {
        SCODE ( MCT *QueryInterface )(
15          IModule *This,
            /* [in] */ REFIID Iid,
            /* [out] */ void **ppObject);

        UINT ( MCT *AddRef )(
20          IModule *This);

        UINT ( MCT *Release )(
            IModule *This);

25      SCODE ( MCT *GetFunctionAddress )(
            IModule *This,
            CTSTR Name,
            UINT Ordinal,
            /* [out] */ ADDRESS *pAddress);
30
        SCODE ( MCT *GetName )(
            IModule *This,
            /* [size_is][out] */ TSTR Buffer,
            /* [in] */ UINT BufSize);
```

```
        SCODE ( MCT *GetArgs )(
            IModule *This,
            /* [size_is][out] */ TSTR Buffer,
            /* [in] */ UINT BufSize);

        SCODE ( MCT *GetLocation )(
            IModule *This,
            /* [out] */ ADDRESS *pAddress,
            /* [out] */ UINT *pSize);

        SCODE ( MCT *GetEntryPoint )(
            IModule *This,
            /* [out] */ ADDRESS *pEntry);

    } IModuleVtbl;
#if defined(__cplusplus) && !defined(CINTERFACE)
    interface IModule : public IUnknown
    {
    public:

        virtual SCODE MCT GetFunctionAddress(
            CTSTR Name,
            UINT Ordinal,
            /* [out] */ ADDRESS *pAddress) = 0;

        virtual SCODE MCT GetName(
            /* [size_is][out] */ TSTR Buffer,
            /* [in] */ UINT BufSize) = 0;

        virtual SCODE MCT GetArgs(
            /* [size_is][out] */ TSTR Buffer,
            /* [in] */ UINT BufSize) = 0;
```

```
        virtual SCODE MCT GetLocation(
            /* [out] */ ADDRESS *pAddress,
            /* [out] */ UINT *pSize) = 0;

        virtual SCODE MCT GetEntryPoint(
            /* [out] */ ADDRESS *pEntry) = 0;

        };

#else   /* C style interface */
        interface IModule
        {
         CONST_VTBL struct IModuleVtbl *lpVtbl;
        };
#endif   /* __cplusplus */ /* C style interface */
#if defined(__cplusplus)
}
#endif /* __cplusplus */

/******** proces.h ********/

#if defined(__cplusplus)
extern "C"{
#endif /* __cplusplus */
typedef interface IProcess IProcess;
typedef /* [transmit] */ void *THREAD_ARGUMENT;

#if 0
typedef /* [transmit] */ void ( __stdcall *THREAD_FUNCTION
)(
    THREAD_ARGUMENT arg);
#else
/* Include THREAD_LINKAGE linkage specification in C/C++
declaration */
```

```
typedef void (THREAD_LINKAGE *THREAD_FUNCTION
)(THREAD_ARGUMENT arg);
#endif

typedef struct _MACHINE_INFO {
    UINT32  MachineId;
    UINT32  OEMId;
    UINT    TotalPhysicalMemory;
    UINT    AvailablePhysicalMemory;
    UINT    Reserved[12];
} MACHINE_INFO;

typedef UINT MACHINE_INFO_FLAVOR;
#define MACHINE_KIND_GENERIC ( 0x01 )
typedef struct IProcessVtbl {
     SCODE ( MCT *QueryInterface )(
         IProcess *This,
         /* [in] */ REFIID Iid,
         /* [out] */ void **ppObject);

     UINT ( MCT *AddRef )(
         IProcess *This);

     UINT ( MCT *Release )(
         IProcess *This);

     SCODE ( MCT *CreateThread )(
         IProcess *This,
         THREAD_FUNCTION pStart,
         THREAD_ARGUMENT Arg,
         ADDR_SIZE StackSize,
         void *Reserved,
         /* [out] */ IThread **ppNewThread);
```

```
        SCODE ( MCT *LoadImage )(
            IProcess *This,
            CTSTR pImage,
            CTSTR pArgs,
            /* [out] */ IModule **ppIModule);

        SCODE ( MCT *Init )(
            IProcess *This,
            ADDR_SIZE StackSize,
            void *compat);

        SCODE ( MCT *MachineInfo )(
            IProcess *This,
            MACHINE_INFO_FLAVOR Kind,
            /* [out] */ MACHINE_INFO *pInfo);

    } IProcessVtbl;

#if defined(__cplusplus) && !defined(CINTERFACE)

    interface IProcess : public IUnknown
    {
    public:
        virtual SCODE MCT CreateThread(
            THREAD_FUNCTION pStart,
            THREAD_ARGUMENT Arg,
            ADDR_SIZE StackSize,
            void *Reserved,
            /* [out] */ IThread **ppNewThread) = 0;

        virtual SCODE MCT LoadImage(
            CTSTR pImage,
            CTSTR pArgs,
            /* [out] */ IModule **ppIModule) = 0;
```

```
        virtual SCODE MCT Init(
            ADDR_SIZE StackSize,
            void *compat) = 0;

        virtual SCODE MCT MachineInfo(
            MACHINE_INFO_FLAVOR Kind,
            /* [out] */ MACHINE_INFO *pInfo) = 0;

    };

#else   /* C style interface */

    interface IProcess
    {
     CONST_VTBL struct IProcessVtbl *lpVtbl;
    };

#endif   /* __cplusplus */ /* C style interface */
#if defined(__cplusplus)
}
#endif /* __cplusplus */
/******** thread.h ********/
#if defined(__cplusplus)
extern "C"{
#endif /* __cplusplus */
typedef interface IThread IThread;
typedef struct IThreadVtbl {
    SCODE ( MCT *QueryInterface )(
        IThread *This,
        /* [in] */ REFIID Iid,
        /* [out] */ void **ppObject);

    UINT ( MCT *AddRef )(
```

```
            IThread *This);

        UINT ( MCT *Release )(
            IThread *This);

        SCODE ( MCT *GetProcess )(
            IThread *This,
            /* [out] */ IProcess **ppIProcess);

    } IThreadVtbl;

#if defined(__cplusplus) && !defined(CINTERFACE)

    interface IThread : public IUnknown
    {
    public:
     virtual SCODE MCT GetProcess(
         /* [out] */ IProcess **ppIProcess) = 0;

    };

#else   /* C style interface */

    interface IThread
    {
     CONST_VTBL struct IThreadVtbl *lpVtbl;
    };

#endif  /* __cplusplus */ /* C style interface */
#if defined(__cplusplus)
}
#endif /* __cplusplus */

EXTERN_C ADDRESS RTLCALLTYPE GetPC(void);
```

```
    /*
     * Current object
     */

    EXTERN_C PITHREAD RTLCALLTYPE CurrentThread(void);
    EXTERN_C PIPROCESS RTLCALLTYPE CurrentProcess(void);
    EXTERN_C PINAMESPACE RTLCALLTYPE CurrentNameSpace(void);
    EXTERN_C PIHEAP RTLCALLTYPE ProcessHeap(void); /* same as
    CurrentHeap */
    EXTERN_C TSTR RTLCALLTYPE ProcessImage(void);
    EXTERN_C TSTR RTLCALLTYPE ProcessArgs(void);
    EXTERN_C PIVMVIEW RTLCALLTYPE CurrentVmView(void);

    /* Atomic
     */
    OSRTL(BOOL) AtomicCmpAndSwap(PUINT pTarget, UINT OldVal,
    UINT NewVal);
    OSRTL(UINT) AtomicAdd(PUINT pTarget, UINT Cnt);
    OSRTL(UINT) AtomicSub(PUINT pTarget, UINT Cnt);
    OSRTL(UINT) AtomicDec(PUINT pTarget);
    OSRTL(UINT) AtomicInc(PUINT pTarget);
    OSRTL(UINT) AtomicSwap(PUINT pTarget, UINT NewVal);
    OSRTL(void) AtomicLIFOInsert(PTR * RESTRICT pHead, PTR
    RESTRICT Item);
    OSRTL(PTR) AtomicLIFORemove(PTR *pHead);

    /* Mutex
     */
    OSRTL(void) Mutex_Init(PMUTEX pmx);
    OSRTL(void) Mutex_Destroy(PMUTEX pmx);
    OSRTL(void) Mutex_Lock(PMUTEX pmx);
    OSRTL(void) Mutex_Unlock(PMUTEX pmx);
    OSRTL(BOOL) Mutex_TryLock(PMUTEX pmx);
```

```
OSRTL(BOOL) Mutex_Locked(const MUTEX *pmx);

/* Condition
 */
OSRTL(void) Condition_Init(PCONDITION pcnd);
OSRTL(void) Condition_Destroy(PCONDITION pcnd);
OSRTL(void) Condition_Signal(PCONDITION pcnd);
OSRTL(BOOL) Condition_InterruptSignal(PCONDITION pcnd);
OSRTL(void) Condition_Broadcast(PCONDITION pcnd);
OSRTL(void) Condition_Wait(PCONDITION pcnd, PMUTEX pmx);
OSRTL(SCODE) Condition_TimedWait(PCONDITION pcnd, PMUTEX
pmx, TIME tout);
OSRTL(SCODE) Condition_WaitAndBeginConstraint(PCONDITION
pcnd, PMUTEX pmx, BOOL endprev, PTIME_CONSTRAINT pcval,
PTIME pttaken);
OSRTL(SCODE)
Condition_TimedWaitAndBeginConstraint(PCONDITION pcnd,
PMUTEX pmx, TIME until, BOOL endprev, PTIME_CONSTRAINT
pcval, PTIME pttaken);

/* Constraint
 */
OSRTL(SCODE) BeginConstraint(BOOL endprev, PTIME_CONSTRAINT
pcval, PTIME pttaken);
OSRTL(SCODE) EndConstraint(PTIME pttaken);

/* Module
 */
OSRTL(SCODE) GetModuleNext(PIMODULE Previous, PIMODULE
*ppIModule);
OSRTL(SCODE) GetModuleContainsAddr(ADDRESS Addr, PIMODULE
*ppIModule);
#define GetCurrentModule(_ppIModule_)
GetModuleContainsAddr(GetPC(), _ppIModule_)
```

```
    /* Compare two UUIDs.  Returns TRUE if they are same, FALSE
    otherwise.
     */
 5  OSRTL(BOOL) UuidCmp(const UUID *pUuid1, const UUID *pUuid2);
    #define IidCmp(pIid1,pIid2)
    UuidCmp((PUUID)pIid1,(PUUID)pIid2)
    #define IpidCmp(pIpid1,pIpid2)
    UuidCmp((PUUID)pIpid1,(PUUID)pIpid2)
10
    /* Various constructors.
     */
    OSRTL(PIHEAP) HeapCreate(ADDRESS Mem, UINT InitSize, UINT
    MaxSize, UINT Flags);
15  OSRTL(IHeap*) CreateHeap(UINT Flags, ADDR_SIZE InitialSize,
    ADDR_SIZE MaxSize);

    /* Others
     */
20  EXTERN_C SCODE WINAPI GetLastError(void);
    EXTERN_C void WINAPI SetLastError(SCODE Error);

    OSRTL(SCODE) AddDevice( PTR Device, PTR Isr, UINT Arg0, UINT
    Arg1, UINT Arg2);
25  #define RemoveDevice(_dev_,_irq_)
    AddDevice(_dev_,NULL,0,_irq_,0)
    OSRTL(SCODE) SleepUntil(TIME Until);
    OSRTL(void)  Delay(UINT MicroSeconds);
    OSRTL(SCODE) GenericQueryInterface(IUnknown *pThis, REFIID
30  IidQuery, void **ppObject, REFIID IidInterface);
    EXTERN_C void CRTAPI WaitFor(IUnknown *pUnk);


    OSRTL(void) ThreadExit(void);
```

```
OSRTL(TIME) CurrentTime(void);

EXTERN_C int CRTAPI memcmp(const void *, const void *,
unsigned);
EXTERN_C void * CRTAPI memcpy(void *, const void *,
unsigned);
EXTERN_C void * CRTAPI memset(void *, int, unsigned);
EXTERN_C void * CRTAPI memmove(void *, const void *,
unsigned);

EXTERN_C size_t CRTAPI strlen(const char *);
EXTERN_C int CRTAPI strcmp(const char *, const char *);
EXTERN_C char *CRTAPI strcpy(char *, const char *);
EXTERN_C char *CRTAPI strcat(char *, const char *);

#endif /* _MMBASE_H_ */
```

APPENDIX E:

The following is a file for reading the foregoing header files of Appendices A-D.

Explanations are provided with the code.

```
------------
#if defined(__cplusplus)
extern "C" {
#endif /* __cplusplus */
This file is common to two programming languages, C and
C++.  Ignore
this clause.

typedef interface ILoadableIPC ILoadableIPC;
Announce that we will define an ILoadableIPC interface,
later on.
```

```
typedef struct ILoadableIPCVtbl {
This is the VTable definition for our interface, it is
only used in
the C language.  But it
is exposed to C++ users as well, in case they need it.

SCODE ( MCT *QueryInterface )(
ILoadableIPC *This,
/* [in] */ REFIID Iid,
/* [out] */ void **ppObject);
This is the standard QueryInterface method, as required
by the base
IUnknown interface.

UINT ( MCT *AddRef )(
ILoadableIPC *This);
Ditto for AddRef()

UINT ( MCT *Release )(
ILoadableIPC *This);
..and for Release().

SCODE ( MCT *TrapHandler )(
ILoadableIPC *This,
PCXTINFO pThreadState
);
We are done with the IUnknown, this method is now
specific to our
interface definition.

} ILoadableIPCVtbl;
End of the VTable definition.
```

```
#if defined(__cplusplus) && !defined(CINTERFACE)
The following defines the ILoadableIPC interface for
users of the
C++ language
interface ILoadableIPC : public IUnknown
In C++ we just say that we "derive" from the IUnknown and
the
compiler will define
the basic methods for us, automatically.
{
public:
virtual SCODE MCT TrapHandler(
PCXTINFO pThreadState
) = 0;
We just define our specific method.  Infact, this portion
between
#if and #else is
the only one that is strictly needed for a C++ uses.
Most of the
stuff in here is for
the benefit of the C programmer.
};

#else  /* __cplusplus /  / C style interface */

interface ILoadableIPC
{
CONST_VTBL struct ILoadableIPCVtbl *lpVtbl;
For the C language we define an interface as something
that just
contains the VTable pointer
to the object's method.  The object's state will actually
follow in
```

```
        memory the lpVtbl pointer.
        };

        #endif  /* __cplusplus / / C style interface */
5
        #if defined(__cplusplus)
        }
        #endif /* __cplusplus */
```